Odyssey 2016
21-24 Jun 2016, Bilbao, Spain

Deep Language: a comprehensive deep learning approach to end-to-end language recognition

Trung Ngo Trong[1,3], *Ville Hautamäki*[1], *Kong Aik Lee*[2]

[1]School of Computing, University of Eastern Finland, Finland

[2]Institute for Infocomm Research, A*STAR, Singapore

[3]Institute of Behavioural Sciences, University of Helsinki, Finland

{trung,villeh}@uef.fi, kalee@i2r.a-star.edu.sg

Abstract

This work explores the use of various Deep Neural Network (DNN) architectures for an end-to-end language identification (LID) task. The approach has been proven to significantly improve the state-of-art in many domains include speech recognition, computer vision and genomics. As an end-to-end system, deep learning removes the burden of hand crafting the feature extraction is conventional approach in LID. This versatility is achieved by training a very deep network to learn distributed representations of speech features with multiple levels of abstraction. In this paper, we show that an end-to-end deep learning system can be used to recognize language from speech utterances with various lengths. Our results show that a combination of three deep architectures: feed-forward network, convolutional network and recurrent network can achieve the best performance compared to other network designs. Additionally, we compare our network performance to state-of-the-art BNF-based i-vector system on NIST 2015 Language Recognition Evaluation corpus. Key to our approach is that we effectively address computational and regularization issues into the network structure to build deeper architecture compare to any previous DNN approaches to language recognition task.

1. Introduction

Spoken language identification (LID) [1] is the process of identifying the language spoken in test utterance. Ideally, the speech segment is assumed to contain only one language, and the recording environment is different among utterances, which can be phone call, interview or public speech with diverse ambient noises. The task attracts increasing attention in the speech community because of potential real world applications. One typical application of LID is language-oriented user interaction, the system acts as a gateway to the service, recognizes user language preference and intelligently customizes the interface [2]. A more challenging task is language diarization [3] which segments and tags a speech utterances based on its language information. This system not only enhances the performance of other speech processing systems include speaker identification and automatic speech recognition (ASR), but also advance the development of universal communication system which can classify and process multilingual audio data.

Many state-of-the-art LID systems still heavily rely on acoustic modeling [4], which requires building a pipeline of handcrafted feature extraction and applicable classifier. One drawback of this design is that the feature representation might not be optimized for the classification objective. Taken into account the issue, deep neural network (DNN) [5], which is a computational model, composes multiple nonlinear layers to capture the complex representations of data with multiple levels of abstraction. The impressive performance of DNN achieved in many domains included image recognition, genomics, and especially ASR [6] has motivated similar approaches to LID [7, 8]. In practice, there are two major deep learning approaches to LID. The first one is "indirect" approach introduced in [7] using deep bottleneck features (BNF) to extract frame-level features for i-vector systems. This approach has proven to give state-of-the-art results in both speaker and language recognition [7].

On the other hand, the "direct" approach constructs an end-to-end classifiers which are trained using spectral information. End-to-end learning allows the network optimized to handle wide range of speech diversity including ambient noise, speakers' variation and recording devices. In [8], it was found that a deep learning system surpassed i-vector based approaches with lower number of parameters when large amount of training data was available. However, the paper only stop at using a single network architecture. Conversely, it was reported in [9] that a combination of many deep architectures outperform conventional deep learning approach to ASR.

In this study, we present the first large scale analysis of various DNN architectures for LID tasks. The key to our approach is *recurrent architecture* of DNN, a model has recently been shown to outperform the state-of-the art DNN systems for acoustic modeling in speech domain [6, 10]. The central idea behind recurrent neural network (RNN) is its feedback connection which creates an internal state to model temporal dependency in data which is essential in speech. Inspired by the work in [9], we conduct a series of experiments using NIST LRE'15 corpus for a systematic study of the best network design for LID task. Our final network is compared to the most recent state-of-the-art BNF i-vector on the same dataset. The results indicate a new potential approach to LID and open a field of research to tackle the challenge of end-to-end LID.

2. Recurrent neural network

A recurrent neural network (RNN) is a variant of artificial neural network where connections between hidden units form a directed cycle. For every time-step t, RNN combines the input vector $\boldsymbol{x}_t$ with their state vector $\boldsymbol{h}_{t-1}$ to produce a next state vector $\boldsymbol{h}_t$ by using a learn-able function with parameters θ.

$$\boldsymbol{h}_t = F(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \theta) \quad (1)$$

doi: 10.21437/Odyssey.2016-16

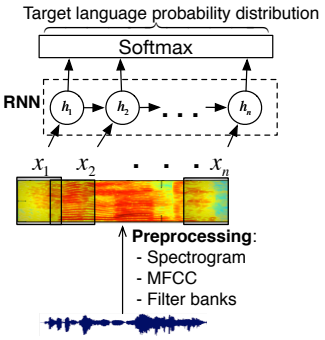

Figure 1: *Design for an end-to-end LID system using RNN.*

This internal state allows it to exhibit dynamic temporal patterns and process arbitrary sequences of inputs. As the network structure reflects strong characteristic of speech signal, it has been widely introduced into speech recognition fields with state-of-the-art performance [10, 11, 12].

2.1. End-to-end RNN for LID

A general approach to end-to-end RNN for language recognition system is described in Figure 1. The system contains three main modules. First, the features preprocessor extracts representative features from raw signal. An RNN network encodes a sequence of speech frames into its hidden states. Finally, a probability decoder projects hidden states of RNN into interpretable probability vector of target languages by using softmax activation function,

$$\varphi(\boldsymbol{y})_j = \frac{e^{\boldsymbol{y}_j}}{\sum_{k=1}^{K} e^{\boldsymbol{y}_k}} \tag{2}$$

where K is the total number of classes, vector $\boldsymbol{y}_j$ is affine transform of stacked RNN's hidden states, and $\varphi(\boldsymbol{y})$ is posterior distribution of target languages. A hard decision can be made by selecting the most probable class. Moreover, another approach transforms the posterior probability into log-likelihood ratio (LLR), which allows more flexible decision making process.

2.2. Tackling long range dependency

Ordinary RNN has convergence issues. In practice, training the network often confront the problem of vanishing gradient and exploding gradient problems as described in [13]. Several architectures were proposed to address this issue. One of the most popular variants uses Gate units to control information flow into or out from the internal state. This is well-known as long-short term memory (LSTM) recurrent network [14]. The key to LSTM is memory cell which is regulated by gating units to update its state over time. As a result, LSTM networks are capable of learning long-term dependencies, and was proven to work tremendously well on a large variety of tasks [14].

The modern architecture of LSTM is illustrated in Figure 2. The whole process can be interpreted as a flow of information vectors from left to right, which includes:

- $\boldsymbol{X}_t$: input vector at time-step t.
- $\boldsymbol{h}_{t-1}$: vector represents previous hidden state at time-step $t-1$.

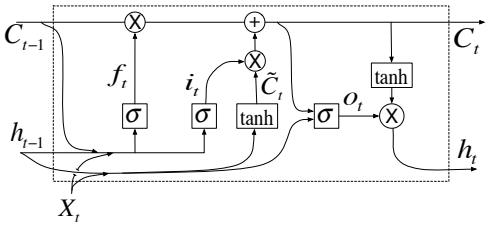

Figure 2: *A LSTM architecture, as a flow of information through memory block which controlled by input gate $\boldsymbol{i}_t$, forget gate $\boldsymbol{f}_t$ and output gate $\boldsymbol{o}_t$*

- $\boldsymbol{c}_{t-1}$: previous memory cell state from time-step $t-1$ encoded as a vector.

The cell state acts like a "conveyor belt", it runs straight down the entire information chain to create precise timing signal, also known as peephole [14]. Additionally, the three vectors form 3 gating units as a "throttle" of information, these units regulate data vectors allowing modification of cell state to capture long-term temporal patterns. The modification includes: store (i.e input gate $\boldsymbol{i}_t$), remove (i.e forget gate $\boldsymbol{f}_t$) and response (i.e output gate $\boldsymbol{o}_t$),

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma_i(\boldsymbol{x}_t\boldsymbol{W}_{xi} + \boldsymbol{h}_{t-1}\boldsymbol{W}_{hi} + \boldsymbol{w}_{ci} \odot \boldsymbol{c}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \sigma_f(\boldsymbol{x}_t\boldsymbol{W}_{xf} + \boldsymbol{h}_{t-1}\boldsymbol{W}_{hf} + \boldsymbol{w}_{cf} \odot \boldsymbol{c}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \sigma_c(\boldsymbol{x}_t\boldsymbol{W}_{xc} + \boldsymbol{h}_{t-1}\boldsymbol{W}_{hc} + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma_o(\boldsymbol{x}_t\boldsymbol{W}_{xo} + \boldsymbol{h}_{t-1}\boldsymbol{W}_{ho} + \boldsymbol{w}_{co} \odot \boldsymbol{c}_t + \boldsymbol{b}_o) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \sigma_h(\boldsymbol{c}_t)
\end{aligned} \tag{3}$$

where $\odot$ represents element-wise operator, and $\boldsymbol{W}_{\_}$ denotes weights matrices (e.g $\boldsymbol{W}_{xi}$ is the matrix of parameters mapping input $\boldsymbol{x}_t$ to input gate dimension). The $\boldsymbol{b}_{\_}$ term denotes bias vectors, σ is activation functions which often are $\mathrm{sigmoid}$ for gate units and $\tanh$ for hidden activation.

According to [14], many variants of LSTM have been proposed since its inception in 1995. Each with their own merits and drawbacks perform differently in various tasks, however, the most remarkable variant is the Gated Recurrent Unit (GRU) [15], it simplifies the LSTM architecture by coupling the input and the forget gate into update gate ($\boldsymbol{u}_t$), together with reset gate ($\boldsymbol{r}_t$) to schedule the update of hidden state. The performance of GRU can be comparable to LSTM, however, its design significantly reduces the number of parameters to be estimated, as follows:

$$\begin{aligned}
\boldsymbol{r}_t &= \sigma_r(\boldsymbol{x}_t\boldsymbol{W}_{xr} + \boldsymbol{h}_{t-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r) \\
\boldsymbol{u}_t &= \sigma_u(\boldsymbol{x}_t\boldsymbol{W}_{xu} + \boldsymbol{h}_{t-1}\boldsymbol{W}_{hu} + \boldsymbol{b}_u) \\
\boldsymbol{c}_t &= \sigma_c(\boldsymbol{x}_t\boldsymbol{W}_{xc} + \boldsymbol{r}_t \odot (\boldsymbol{h}_{t-1}\boldsymbol{W}_{hc}) + \boldsymbol{b}_c) \\
\boldsymbol{h}_t &= (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{u}_t \odot \boldsymbol{c}_t
\end{aligned} \tag{4}$$

In this paper, we investigate the use of both LSTM and GRU to select the best architecture for language recognition task.

3. Language identification corpus

In this study, we experiment with NIST LRE15 training and evaluation corpora. We decided to leave out French cluster as it has inconsistency between training and evaluation partitions. [1]

[1] In NIST LRE15 workshop, NIST also excluded the French cluster from the analysis of results

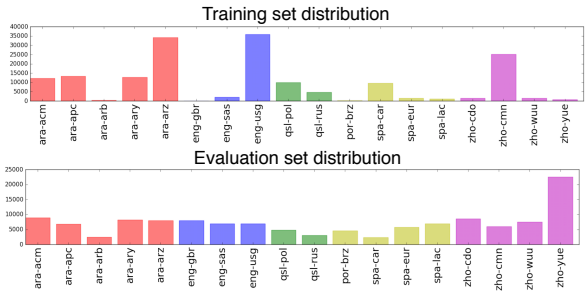

Figure 3: *Distribution of each language in training set and evaluation set*

3.1. Data distribution

The dataset contains ≈ 796 hours of speech. The speech was recorded in various conditions, from phone conversation with ambient noise to a formal interview. There is large difference between the language distribution of training and evaluation data as shown in Figure 3. Furthermore, our inspection on three random speech utterances from 3 clusters suggests diverse structures in the audio clips, some utterances contain very long silences mixed with noise and the speech activities are occasionally short and meaningless.

This observation implies the importance of robustness in learning speech utterances representation, a distributed representation obtained by deep network can learn many intermediate concepts that are useful to capture the statistical dependencies of input signal and output language. Since the primary cost function of NIST LRE' 15 is applied separately for each language cluster [16], we decided to train different network for each cluster, and the final C_{avg} is average of all clusters' scores.

3.2. Performance measurement

Output of our system is a (20 dimensional) vector of log likelihood ratio (LLR) scores for each test segment. The objectives of LRE15 is minimizing the criterion in Eq. (5), which apply for each cluster and its pairs of target/non-target languages (L_T, L_N):

$$C_{\text{avg}} = \frac{1}{N_L}\{[C_{\text{miss}} * P_{\text{Target}} * \sum_{L_T} P_{\text{Miss}}(L_T)] + \frac{1}{N_L - 1} [C_{\text{FA}} * (1 - P_{\text{Target}}) * \sum_{L_T}\sum_{L_N} P_{\text{FA}}(L_T, L_N)]\} \tag{5}$$

where N_L is the number of language in the cluster, C_{miss}, C_{FA} and P_{Target} are application-specific parameters represent the weights of detection miss and false alarm probabilities. For LRE15, the application parameters will be: $C_{\text{miss}} = C_{\text{FA}} = 1$, and $P_{\text{Target}} = 0.5$. Additionally, the average of this value across six clusters will be the criterion to judge final performance of our system.

In practice, we trained our model using cross-entropy loss (Eq. (6) to categorize each training frames to its true language label.

$$L = -\sum_i t_i \log(p_i) \tag{6}$$

where t_i is one-hot-encoded true class label, and p_i is our predicted probabilities. Acknowledge the nontrivial difference between distribution of training and evaluating dataset, we propose a dataset partition strategy specified in Table 1. Note that all dataset are randomly splitted and completely non-overlap.

Table 1: *Splitting scheme for training and evaluation corpora*

Alias	Corpus	Partition	Purpose
Set01	Dev.	20%	Validating training results on training dataset
Set02	Dev.	80%	all networks were trained using this data set
Set03	Eval.	20%	Tuning, features selection, model selection
Set04	Eval.	80%	Evaluate overall performance of system

We also use accuracy score (Eq. (7)) as our criterion for Section 3.3 and 4, because the measurement is faster to calculate and the value also reflect actual LRE15 evaluation score

$$\frac{\sum_i \mathbb{I}(t_i = \text{argmax}(p_i))}{n}, \tag{7}$$

where: $\mathbb{I}$ is identity function and n is the number of samples. It is notable that we compute this score for each example and one utterance can be segmented into many examples with the same label. Further study in Section 5 and 6 will average softmax probabilities from all samples of each utterance as final score, and C_{avg} on these scores are used for evaluation as we need more precise criterion to compare our network to baseline i-vector approach.

3.3. Feature selection

In all of our experiments, the audio data was preprocessed into 25 ms frames, overlapped by 10 ms. The extracted coefficients were individually normalized using local mean and standard deviation of each utterance to be centered at 0 with variance equal to 1. For feature selection, we separately train a network for each feature configuration, the network has 3 stacked LSTM layers with 250 units per layer.

We initially use 40 filter banks in log-Mel scale feature and propose 4 schemes to process speech utterance into relevant feature vectors. The length of input sequences is critical parameters, as a sufficiently long input should contain all important temporal patterns for RNN. Consequently, frames from each utterance are grouped into fixed-length sequences, and the utterances, which are shorter, are padded by zeros. We also introduce masks (i.e indicator vectors of 0 and 1, 0 - for not used in training frames and 1 - otherwise) for each training examples to carefully leave out these padded frames during recurrent steps. Subsequently, VAD can be used inclusively with mask indices, however, our approach using Signal to Noise Ratio threshold cannot exclude long silences, so we didn't introduce VAD in our experiments.

Table 2: *Accuracy (Eq. (7)) on Set03 using different schemes*

Length (Frames)	Overlap (Frames)	zho	qsl	spa	Avg.
20	0	0.32	0.68	0.35	0.45
500	0	**0.39**	**0.74**	**0.53**	**0.557**
500	100	0.38	0.73	**0.53**	0.547
800	0	0.32	0.67	0.49	0.493

For short input sequences (i.e. 20 frames), we miss important temporal information, and there exist cases that the input

only contain long silent signal which bias the network to wrong structure. Furthermore, longer sequences (i.e. 800 frames) also hurt the overall performance. Consequently, we use the second configuration in Table 2 for further experiments.

Given that the most popular features to train DNNs and their variants are log-Mel filter bank features, some ASR systems used Mel-frequency cepstral coefficients (MFCCs) features and achieved reasonable results on NIST LRE 2009 dataset [6]. In our experiment, the average performance in Table 3 indicates more advantages of using filter bank features. As we are going to add convolutional layers, log-Mel features are known to be more friendly to the convolution operators [17, 18]. The convolutional layers will reduce the spectral variation and model the correlation among different frequency banks. Hence, for the rest of the paper, we use log-Mel filter banks features rolled into sequences of 500 frames without overlap.

Table 3: *Accuracy on Set03 using different type of features*

Features	zho	qsl	spa	Avg.
MFCC	**0.40**	0.68	0.51	0.530
Logmel filter banks	0.39	**0.74**	**0.53**	**0.557**

4. Augmenting Network Architecture

In this section, we investigate a relevant deep learning approach to LID. The optimal design is selected by planning experiments to augment the network structure. We start with the proposed architecture with 1 LSTM of 512 memory cells in [8] as the baseline for augmenting the network.

All networks are initialized using the same random seed to remove randomness on the results. Unless otherwise indicated, we use 40 dimensional log-Mel filter-banks + delta + double delta coefficients to train our networks. Furthermore, each speech utterances' features are rolled into many sequences of 500 frames without any overlap as specified in Section 3.2.

To optimize the network, we use Rmsprop optimizer [19] (Eq. (8)), the algorithm scale the learning rates ($\boldsymbol{\eta}$) by dividing with the moving average of the root mean squared gradients which controlled by a decay factor (ρ). Hence, the learning rate is adapted to current convergent speed of the network,

$$\begin{aligned} \boldsymbol{r}_t &= \rho \boldsymbol{r}_{t-1} + (1-\rho) * \boldsymbol{g}^2 \\ \boldsymbol{\eta}_t &= \frac{\boldsymbol{\eta}}{\sqrt{\boldsymbol{r}_t + \epsilon}}, \end{aligned} \tag{8}$$

where $\boldsymbol{g}$ is gradients' matrix of network parameters. We choose $\rho = 0.9$ and $\boldsymbol{\eta} = 0.0001 - 0.001$ depend on the number of parameters, the number of layers of the network and our experiments. All the networks are first trained to convergence with batch size 128 and dropout ($p = 0.5$) enabled, then we do fine-tuning by adding Gaussian noise to activation of each layer to slightly perturb the network converge to a better region.

4.1. The power of depth network

As suggested in [20], a thin deep recurrent neural network can significantly outperform shallow version with the same number of parameters. It is also mentioned in [21] that there are simple functions expressible by small 3-layer feed-forward neural networks which cannot be approximated by a 2-layer network. Furthermore, our results in Table 4 also emphasize the importance of depth when constructing neural network. The first network with depth one is our baseline LSTM. The next two network designs contain only 250 units per layer, as a result, they have lower number of parameters, but achieved greater performance on most of the clusters.

Table 4: *Accuracy on Set03 with various network depth (units* are the number of units for each layer)*

Depth × units* # Param.	1×512 2.8×10^6	2×250 1.6×10^6	3×250 2.1×10^6
ara	0.44	0.48	**0.48**
eng	0.43	**0.45**	0.43
zho	0.37	0.35	**0.39**
qsl	0.67	0.73	**0.74**
spa	**0.53**	0.49	**0.53**
Avg.	0.488	0.500	**0.514**

4.2. RNN variants

As proposed in Section 2, we conduct our experiments on three different variants of RNN to select the most appropriate architecture for language recognition task. The three variants are vanilla RNN, LSTM and GRU. We use the network design from previous section with 3 layers of 250 hidden units for each layer.

Table 5: *Accuracy on Set03 of 3 RNN variants*

Variants	RNN	LSTM	GRU
ara	-	**0.48**	0.46
eng	-	**0.43**	**0.43**
zho	-	**0.39**	**0.39**
qsl	-	0.74	**0.76**
spa	0.42	**0.53**	0.50
Avg.	-	**0.514**	0.508

The RNN network is difficult to train properly on long sequences, the training took longer time to converge and we have to use gradient clipping with maximum norm of 50 [22]. RNN also result very poor performance on *spa* clusters, and we skip its training process for other clusters since it showed no improvement compared to previous approaches. Conversely, GRU and LSTM converge without gradient clipping and achieve comparable results. For the overall performance, LSTM outperforms GRU with relative 1.2% improvement, however, GRU is more computationally efficient, since LSTM uses 45.8% more parameters but only results in 0.6% improved accuracy. As a result, we use GRU to build deeper architecture due to this merit.

4.3. Multiple architecture design

Feedfoward neural network (FNN), convolutional neural network (CNN), and recurrent neural network (RNN) are complementary in their learning capabilities to capture different patterns. While FNN, with multiple processing layers, is able to extract hierarchical representations that benefit the discriminative objective, CNN has ability to extract local invariant features in both time and frequency domain [17]. Since the learned representation from CNN is heavily relied on internal structure of data, stacking multiple convolutional layers after the input can capture robust low-level features of the signal, and was reported in [9, 17] to boost the network overall performance. We propose an architecture that leverages the merits of three different variants.

Our first two layers are convolutional layers with 128 feature maps for each layer. The first layer convolve both in time

and frequency domain with filter size of 9×9 to extract invariant local representation of spectral information. The second layer use 3×5 filter with 1×2 stride to keep time dimension unchanged. A pooling size of 3 on frequency axis was used for the first layer, and no pooling was done in the second layer.

The dimension of the last layer of our CNN is large, because of the increasing in number of feature maps. As suggested in [9], adding a linear layer to perform dimensional reduction provides more compact representation without any accuracy tradeoff. Therefore, we form a linear projection to map output of CNN to 256 dimensions before feeding these features to the next 2 GRU layers of 250 units each.

In the final state, we add 1 fully connected layer of size 512 before the softmax layer. Our network has depth of 5 layers, as the depth increase we need strong regularization methods. We adopted two well-known techniques dropout [23] and batch normalization [24]. However, implementing batch normalization effectively for RNN is a difficult task as we have a sequence of input, we only introduce batch normalization to the first 2 convolutional layers and compare its result to the dropout version of the network.

Table 6: *Accuracy on Set03 of 3 design*

	LSTM	CGFNN^1	CGFNN^2
ara	0.48	**0.49**	**0.49**
eng	0.43	0.44	**0.55**
zho	0.39	0.43	**0.51**
qsl	0.74	0.78	**0.84**
spa	0.53	**0.57**	0.53
Avg.	0.514	0.542	**0.584**

CGFNN is our proposed architecture, a combination of CNN, GRU and FNN. CGFNN^1 is the design using dropout for convolution layers, vice versa, CGFNN^2 is same design but using batch normalization. The results indicate improvement by using combined architecture, and batch normalization has proved its indisputable efficiency in regularizing CNN. One of the feasible explanations is that the gradients of convolution layers are averaged over the spatial extent of the feature maps. Since dropout stochastically removes activation, hence, their gradients, the backpropagation end up having many correlated terms in the averaged gradient, each with different dropout patterns. As a results, the network converges slower and the learned local patterns become unstable.

4.4. Recurrent pooling in time

As GRU returns a full sequence of 250 frames rolling in time (sub-sampled from 500 to 250 by CNN), the dimension is enormous (i.e $250 * 250 = 62500$). A projection from RNN output to feed-forward layer can have $32 * 10^6$ parameters which consumes huge amount of memory and computational resources.

In [25], the authors argue that such high dimensional representation is overly precise and contain much redundant information, and they suggest a pooling over time strategy which is illustrated in Figure 4. The pooling modules can be max or average pooling, however, they have different interpretations when we apply pooling for time axis. Therefore, we perform an empirical comparison of two approaches on overall performance. The tests are performed on 3 clusters: *zho*, *qsl* and *spa* because of the computational cost.

Table 7 shows promising result of using average pooling over max pooling. A reasonable explanation is that obtaining

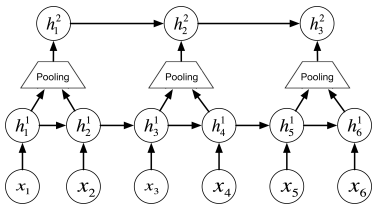

Figure 4: *A pooling over time dimension of RNN*

Table 7: *Accuracy on Set03 of different pooling function*

	GRU	avr. pooling	max pooling
zho	0.39	**0.43**	0.31
qsl	**0.76**	**0.76**	0.62
spa	**0.50**	0.49	0.49

an invariant representation by selecting maximum between 2 high-dimensional vectors of 250 dimensions is difficult, hence, the network drops its temporal information after every pooling step. Since this issue doesn't appear to average pooling, it is good practice to add time pooling layer before we project RNN output to fully connected layers.

In summary, the final architecture is a combination of CNN, RNN, FNN with addition of batch normalization for CNN and time pooling for RNN. As we scale up the network for NIST LRE'15, we will further investigate deeper architecture based on these analysis in the next section.

5. Deep Language Network

5.1. Network design

As the number of GRU layers increase, placing time pooling after each layer becomes inefficient, we are confronted performance degradation as we continue training. Since pooling is a dimension reduction technique, this means an amount of information along time axis is diminished after each GRU layer. To address this issue, we only do pooling in time for the last two GRU layers. Since the first two layers are responsible for learning more primitive and robust representation, dropping frames at higher layers forces the network to learn more compact and abstract features. As a result, this strategy reduces the number of projection parameters by 4 times without performance tradeoff.

Regardless a small amount of parameters added by introducing convolutional layers, the size of the tensor during convolutional computation is multiplied by the number of feature maps which consumes significant amount of memory. Hence, we decided to keep 128 feature maps for each CNN layer which leads to our final architecture illustrated in Figure 5.

5.2. Training

Generally, we apply the same training strategy in Section 4. However, as the network scales up, we make some modifications to the strategy. We choose appropriate learning rate by Eq. (9), the equation adapts the number of parameters and the depth of network to choose an initial learning rate that guarantees the network will converge at a reasonable speed.

$$\begin{aligned} \lambda &= \mathrm{nparams} * \sqrt{\mathrm{nlayers}} \\ \eta &= 10^{\log_{10}\left(\frac{1}{\lambda^{1/2.03}}\right)} \end{aligned} \tag{9}$$

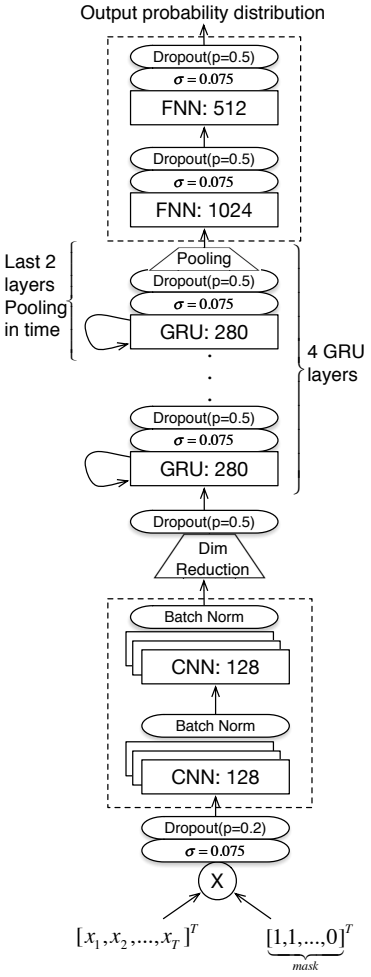

Figure 5: *Deep Language: deep neural network for language recognition*

where λ is an estimation of network complexity, the value **2.03** is a heuristic value, if this value is greater than 2 the initial learning step increase significantly, however, it increase the risk that the gradients might explode or vanish during training. Our observation showed that a sufficiently small initial learning rate is more important for adaptive optimization algorithm like rmsprop [19], especially for deep RNN network, as the gradients rolled back in time, they grow very fast for the first few steps.

The network is first trained to converge in 10 epochs with dropout enabled and Gaussian noise turned off. Conversely, the fine tuning process turns on Gaussian noise without dropout to perturb the weights for better generalization. We use generalization loss (GL) as early stopping criterion [26] and decrease learning rate by 1.5 whenever the network drops its tuning score.

The regularizing effect of L2-norm isn't clear for small network. However, we figure out that including L2 regularization for training a big network not only improve the result, but also speed up the convergence, since it constraints the weights in acceptable range and doesn't allow the optimizing process go too far from the best generalization region. Unlike high probability dropout, adding small fraction of dropout before the input to convolutional layers helps CNN learn more robust local representation of spectrogram.

5.3. Compare to previous DNN

We compare Deep Language to $CGFNN^2$ network specified in Section 4.3. The overall performance was improved by using deeper architecture. It is notable that keep increasing the depth exposes our network to strong over-fitting, hence, we stop augmenting the network and compare proposed model with the BNF baseline.

Table 8: C_{avg} *on Set04 of* $CGFNN^2$ *and Deep Language*

	CGFNN^2	DeepLang
ara	**30.16**	30.51
eng	**32.84**	32.95
zho	29.87	**29.39**
qsl	19.23	**18.95**
spa	39.72	**36.38**
Avg.	30.364	**29.636**

6. Experiments

6.1. Bottleneck feature based classifier

We compare our system performance on the bottleneck DNN (BNF) feature [7]. We represent each stream of BNF's in one utterance by an i-vector. Final classification is performed by *multi-class logistic regression* (MCLR).

A bottleneck DNN was trained using the 40-dimensional filter bank features with the first and second order derivatives extracted from the switchboard landline data. The features were then applied a global mean and variance normalization followed by a per utterance mean and variance normalization before feeding to the DNN. Random weight initialization is used to start the DNN training. The DNN input contains 21 stacked frames rending an input layer with 2520 units. Seven hidden layers including one bottleneck layer were trained. Each hidden layer except the bottleneck layer has 1024 hidden units and uses the rectified linear unit (ReLU) activation function. The second to last hidden layer is the bottleneck layer with 64 output units and linear outputs are extracted as the bottleneck features. The output layer has 6111 units corresponding to 6111 senones obtained from the baseline speaker-independent GMM-HMM system trained with 39-dimensional MFCC features (13 static features plus first and second order derivatives) extracted from the switchboard landline data.

The 64-dimensional bottleneck features are used for extracting the i-vectors. An energy-based voice activity detection (VAD) technique was applied to the raw bottleneck features to exclude the silence frames. The voiced frames were then used to train a universal background model with 1024 Gaussians with diagonal covariances. The diagonal UBM was then used as an initial point to train a full-covariance UBM with 1024 Gaussians. The full-covariance UBM is then used to train the total variability matrix and extract the i-vectors.

The MCLR system is based on the multi-class cross-entropy discriminative training in the score vector space. To this end, i-vectors were transformed into log-likelihood score vectors through a set of Gaussian distributions, each representing the distribution of the language class in the i-vector space. As the amount of data is extremely imbalance among classes,

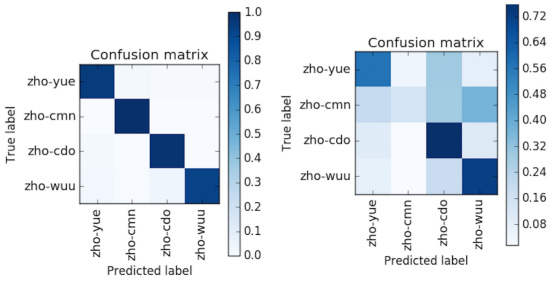

Figure 6: *Confusion matrix of our prediction on zho Set01 (left) and Set04 (right)*

with some languages limited to less than an hour of speech, we trained a global covariance matrix where language-specific covariance could be derived with a smoothing factor of 0.1. Given a test i-vector, a score vector is obtained by concatenating the log-likelihood scores from these Gaussian distribution. Discriminative training is further applied on the score vector.

6.2. Results

Table 9 summarizes the results of the two systems on validation set. We highlight two major results. First, the proposed architecture outperformed BNF i-vector approach on tuning set which has similar distribution as training data. Second, The distribution of C_{avg} is significantly different between 2 approaches, which indicates that the two algorithms exploiting different discriminative information which can benefit complementary tasks.

Table 9: C_{avg} *on Set01, BNF i-vector baseline, Deep Language*

	BNF i-vector	DeepLang
ara	1.23	4.36
eng	1.46	0.26
zho	3.56	2.28
qsl	1.74	1.81
spa	11.01	6.28
Avg.	3.798	2.998

Conversely, our network shows its drawback in generalizing to different data distribution. The performance rapidly drops according to the divergence between the 2 distribution. Table 10 emphasizes the weakness of strong nonlinear model compared to BNF i-vector approach.

Table 10: C_{avg} *on Set04, i-vector baseline, Deep Language*

	BNF i-vector	DeepLang
ara	22.08	30.51
eng	11.52	32.95
zho	16.91	29.39
qsl	6.46	18.95
spa	22.27	36.38
Avg.	15.848	29.636

Our further analysis shows that the degradation effect is mostly created by dominant classes, Figure 6. In the case of *zho* cluster, *zho-cmn* language has $\approx$ 87% of training data,

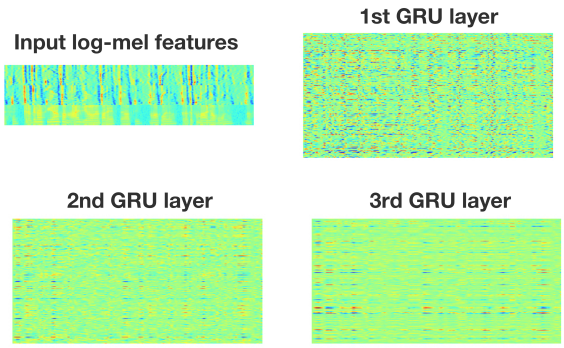

Figure 7: *Visualization of learned representation after 3 GRU layers*

hence, most of the gradients will be backpropagated by examples from this class and the network is optimized to predict the given language with $> 90\%$ accuracy. Contradictory, the side-effect of dominant classes is that they prevent over-fitting of other classes, with lower amount of data, *zho-cdo*, *zho-wuu* and *zho-yue* are thus better generalized.

6.3. Interpretation of learned representation

A distributed temporal representation learned by deep RNN can be visualized by feeding the spectral input to the network and plotting the activation after each layer. We used a simple model of 3 GRU layers with 250 cell units each layer to interpret the learning process of RNN, illustrated in Figure 7.

At a first glance, we can see the time dependency of original spectral data is reserved after each layer. However, the representation is significantly simplified and abstracted after every depth of the network. The first GRU learns a very noisy structure of the spectra, this is low level representation as the network is searching and remembering all fundamental temporal patterns of input. The activation of second layer was smoothed showing that the network is learning more abstract representation by concentrating on stronger temporal dependency of the input signal (i.e the vertical line of the image on bottom left corner). The last layer shows more smooth pattern with clear focus point which is highlighted by the network.

7. Conclusions

In this work, we investigate a comprehensive deep learning approach to end-to-end automatic language identification (LID). Motivated by the recent success of DNN to speech recognition task, we explored a combination of the most advanced network architectures including: CNN, RNN and FNN to replace the pipeline of handcrafted features with BNF and i-vector. Our architecture has taken into account the computational issues and regularization effect to construct deeper network in order to address large scale LID task.

Even though our proposed architecture hasn't surpassed the recent state-of-the-art BNF i-vector system, the trained model shows a very promising results when combine multiple architecture for LID. Our Deep Language system can outperform the shallow and single architecture approach. An initial good result on validation set with a moderate performance on evaluation data suggests that the network was able to capture long-term temporal dependency of speech utterances which is relevant for

language recognition task.

On the other hand, the degradation of the system on evaluation copra leaves plenty of room for further improvement. The authors from [27] emphasizes that imbalanced training data potentially has negative impact on overall performance in deep networks. As a result, the paper suggests oversampling the training dataset to reduce this pessimistic effect. Additionally, one of the reasonable explanations for the bad generalization of the network is the effect of majority class on cross-entropy objective function. Because of hard decision labels, cross-entropy function only backpropagates gradients of target class regardless all other output information. Two feasible solutions are using Bayesian cross-entropy cost function to normalize the errors based on class priors [27], and leveraging large amount of evaluation data by proportionally fitting the model on pseudo-labeled test data [28]. It is also notable that BNF of baseline approach was trained using external dataset (i.e Switchboard corpus). Hence, a strategy, involves pretraining the network to capture the essence of speech, then, fine-tunes it using LRE'15 corpus for language discrimination, might be explored.

8. Acknowledgements

The work is partially supported by the Academy of Finland in the DigiSami-project (Fenno-Ugric Digital Citizens, grant n^{o}270082).

9. References

[1] Haizhou Li, Bin Ma, and Kong Aik Lee, "Spoken language recognition: From fundamentals to practice," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1136–1159, May 2013.

[2] Sara Bongartz, Yucheng Jin, and Paternò el at., "Adaptive user interfaces for smart environments with the support of model-based languages," *Ambient Intelligence: Third International Joint Conference.*, 2012.

[3] D. C. Lyu, E. S. Chng, and H. Li, "Language diarization for code-switch conversational speech," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 7314–7318.

[4] Najim Dehak, Pedro A Torres-Carrasquillo, Douglas A Reynolds, and Reda Dehak, "Language recognition via i-vectors and dimensionality reduction," in *Interspeech*. Citeseer, 2011, pp. 857–860.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 05 2015.

[6] Geoffrey E. Hinton, Li Deng, and Dong Yu el at., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[7] Fred Richardson, Douglas A. Reynolds, and Najim Dehak, "A unified deep neural network for speaker and language recognition," *CoRR*, vol. abs/1504.00923, 2015.

[8] Javier Gonzalez-Dominguez, Ignacio Lopez-Moreno, and Hasim Sak, "Automatic language identification using long short-term memory recurrent neural networks," *Interspeech*, 2014.

[9] T.N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *ICASSP*, April 2015, pp. 4580–4584.

[10] Dario Amodei, Rishita Anubhai, and Eric Battenberg et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015.

[11] Tony Robinson, Mike Hochberg, and Steve Renals, *Automatic Speech and Speaker Recognition: Advanced Topics*, chapter The Use of Recurrent Neural Networks in Continuous Speech Recognition, pp. 233–258, Springer US, Boston, MA, 1996.

[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton, "Speech recognition with deep recurrent neural networks," *ICASSP*, 2013.

[13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, "On the difficulty of training recurrent neural networks," *JMLR*, 2013.

[14] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015.

[15] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[16] "The 2015 nist language recognition evaluation plan," [Online] at: http://www.nist.gov/itl/iad/mig/lre11.cfm.

[17] Tara N Sainath and Kingsbury el at., "Deep Convolutional Neural Networks for Large-scale Speech Tasks," *Neural Networks*, pp. 1–10, Nov. 2014.

[18] O. Abdel-Hamid, A. Mohamed, Hui Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *ICASSP*, March 2012, pp. 4277–4280.

[19] T. Tieleman and G. Hinton, "Neural networks for machine learning, lecture 6.5 - rmsprop," *2012*, https://www.coursera.org/course/neuralnets.

[20] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio, "How to construct deep recurrent neural networks," *CoRR*, vol. abs/1312.6026, 2013.

[21] R. Eldan and O. Shamir, "The Power of Depth for Feedforward Neural Networks," *ArXiv*, Dec. 2015.

[22] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le, "Sequence to sequence learning with neural networks," *NIPS*, 2014.

[23] Nitish Srivastava, Geoffrey Hinton, and Alex Krizhevsky el at., "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, pp. 1929–1958, 2014.

[24] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[25] Dzmitry Bahdanau, Jan Chorowski, and Dmitriy Serdyuk el at., "End-to-end attention-based large vocabulary speech recognition," *CoRR*, vol. abs/1508.04395, 2015.

[26] Lutz Prechelt, *Neural Networks: Tricks of the Trade: Second Edition*, chapter Early Stopping — But When?, pp. 53–67, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[27] Paulina Hensman and David Masko, "The impact of imbalanced training data for convolutional neural networks," *Degree Project in Computer Science, KTH Royal Institute of Technology*, 2015.

[28] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *ArXiv*, Mar. 2015.